



下载APP



01 | 技术概览：eBPF 的发展历程及工作原理

2022-01-17 倪朋飞

《eBPF 核心技术与实战》

课程介绍 >



讲述：山荣

时长 10:15 大小 9.40M



你好，我是倪朋飞。

在正式介绍 eBPF 的使用方法和具体应用之前，我会用两讲的内容，带你了解 eBPF 的技术脉络和学习路线，为你后面的学习做好准备。

在开篇词里，我带你一起了解了这门课的设计思路和主要内容，也简单介绍了 eBPF 的主要应用场景，包括故障诊断、网络优化、安全控制、性能监控等。那你可能就要问了：为什么 eBPF 可以应用到这么广泛的领域呢？



eBPF 广泛的应用场景和强大的功能，跟它的发展历程、基本原理密切相关。那么，eBPF 的发展历程是什么样的？它又是如何在确保安全的前提下，允许非内核开发者去扩展内核的功能的呢？今天，我就带你一起来看看这些问题。

eBPF 的发展历程是什么样的？

在开篇词中，我曾经提到，eBPF 是从 BPF (Berkeley Packet Filter) 技术扩展而来的。而说起 BPF，它的历史就更悠长了。

早在 1992 年的 USENIX 会议上，Steven McCanne 和 Van Jacobson 发布的论文“[The BSD Packet Filter: A New Architecture for User-level Packet Capture](#)”就为 BSD 操作系统带来了革命性的包过滤机制 BSD Packet Filter（简称为 BPF），这比当时最先进的数据包过滤技术还快 20 倍。为什么性能这么好呢？这主要得益于 BPF 的两大设计：

第一，内核态引入一个新的虚拟机，所有指令都在内核虚拟机中运行。

第二，用户态使用 BPF 字节码来定义过滤表达式，然后传递给内核，由内核虚拟机解释执行。

这就使得包过滤可以直接在内核中执行，避免了向用户态复制每个数据包，从而极大提升了包过滤的性能，进而被各大操作系统广泛接受。BPF 最初的名字 BSD Packet Filter，也被作者的工作单位名所替代，变成了 Berkeley Packet Filter（很巧的是，还是简称 BPF）。

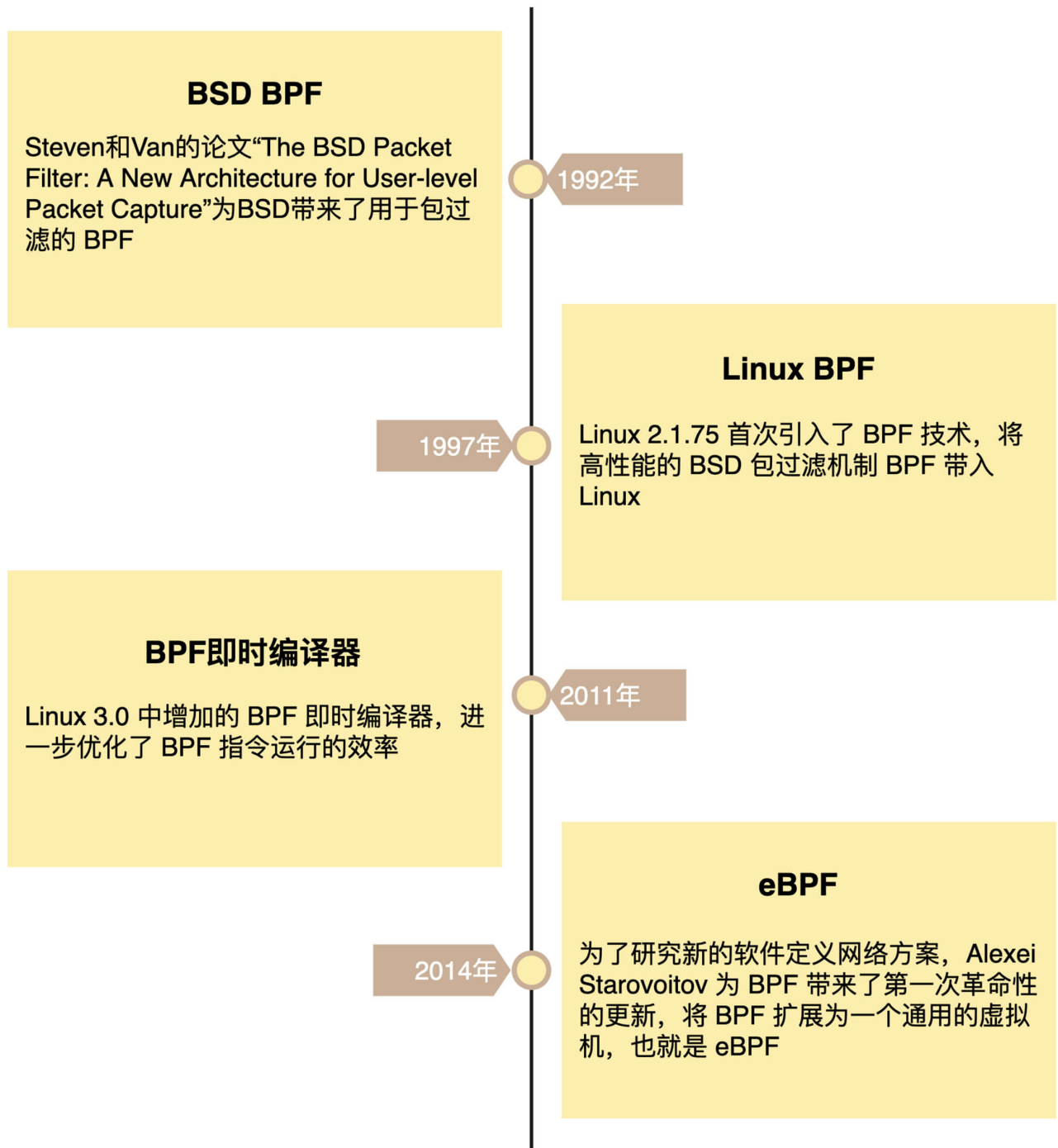
在 BPF 诞生五年后，Linux 2.1.75 首次引入了 BPF 技术，随后 BPF 开始了不温不火的发展历程。其中，Linux 3.0 中增加的 BPF 即时编译器可以算是一个最重大的更新了。它替换掉了原本性能更差的解释器，进一步优化了 BPF 指令运行的效率。但直到此时，BPF 的应用还是仅限于网络包过滤这个传统的领域中。

时间到了 2014 年。为了研究新的软件定义网络方案，Alexei Starovoitov 为 BPF 带来了第一次革命性的更新，将 BPF 扩展为一个通用的虚拟机，也就是 eBPF。eBPF 不仅扩展了寄存器的数量，引入了全新的 BPF 映射存储，还在 4.x 内核中将原本单一的数据包过滤事件逐步扩展到了内核态函数、用户态函数、跟踪点、性能事件（perf_events）以及安全控制等。

eBPF 的诞生是 BPF 技术的一个转折点，使得 BPF 不再仅限于网络栈，而是成为内核的一个顶级子系统。

在内核发展的同时，eBPF 繁荣的生态也进一步促进了 eBPF 的蓬勃发展。这其中，最典型的的就是 iovisor 带来的 BCC、bpfftrace 等工具，成为 eBPF 在跟踪和排错领域的最佳实践。由于 eBPF 无需修改内核源码和重新编译内核就可以扩展内核的功能，Cilium、Kraton、Falco 等一系列基于 eBPF 优化网络 and 安全的开源项目也逐步诞生。并且，越来越多的开源和商业解决方案开始借助 eBPF，优化其网络、安全以及观测的性能。比如，最流行的网络解决方案之一 Calico，就在最近的版本中引入了 [eBPF 数据面网络](#)，大大提升了网络的性能。

为了帮你更好地理解 eBPF 的发展历程，我把 eBPF 诞生以来的发展过程整理成了一张图片：



BCC

BCC (BPF Compiler Collection) 提供了一系列基于 eBPF 的工具和库函数，大大简化了 eBPF 程序的开发和运行。同年，Linux 4.1 也开始支持 kprobe 和 cls_bpf (用于tc)

2015年

XDP和跟踪点

Linux 4.7-4.10 带来了跟踪点、perf 事件、XDP 以及 cgroups 的支持，丰富了 eBPF 的事件源。同年，Cilium 项目发布

2016年

bpftool和libbpf

BPF 成为内核独立子模块，并支持了 kTLS、bpftool、libbpf 等。同年，Netflix、Facebook、Cloudflare 等公司开始将 eBPF 用于跟踪、DDoS 防御、4 层负载均衡等

2017年

BTF

BPF 新增了轻量级调试信息格式 BTF 以及新的 AF_XDP 类型。同年，Cilium 发布 1.0 版本，bpftool 和 bpffiler 项目也正式发布

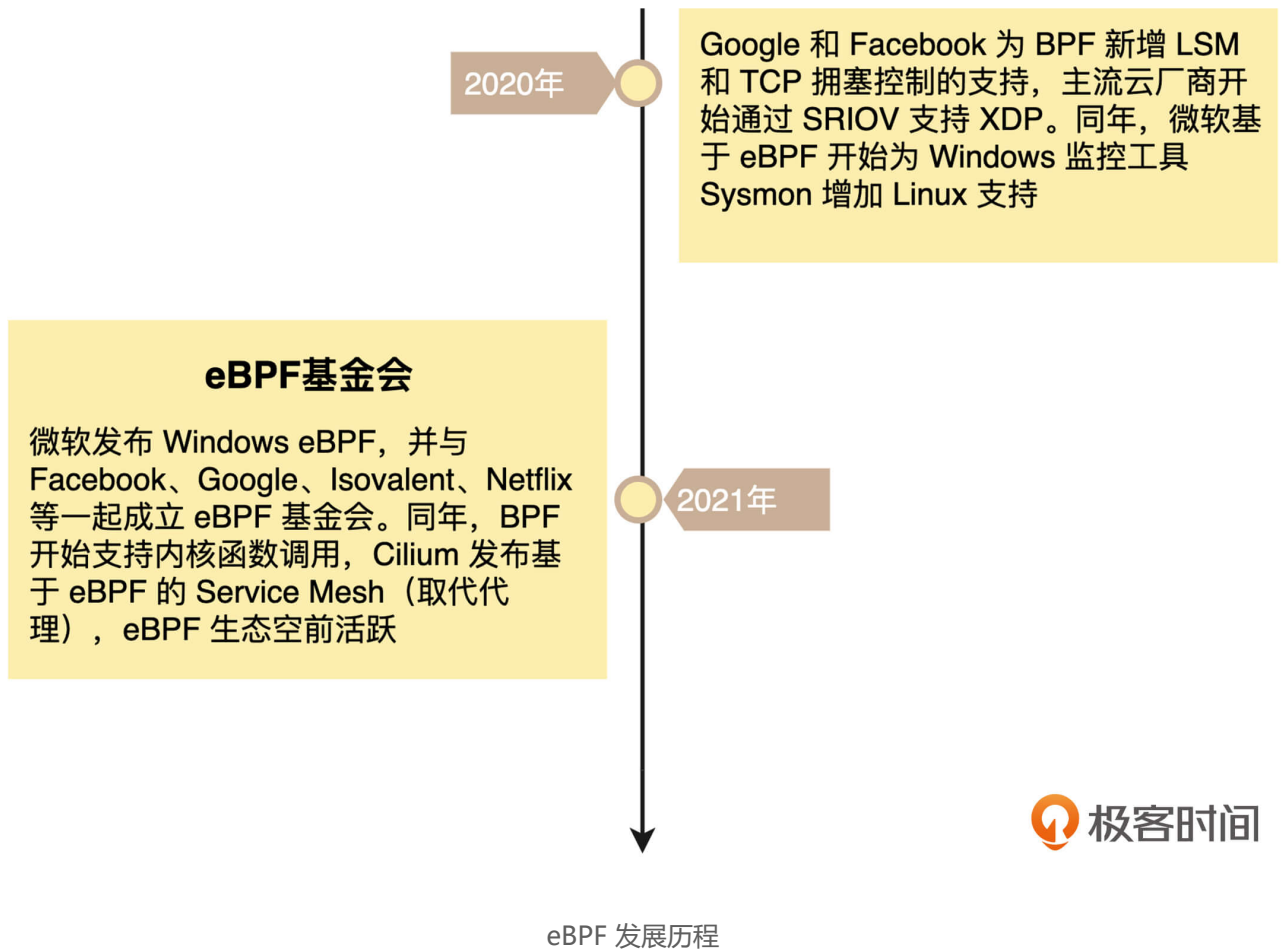
2018年

尾调用和热更新

BPF 新增了尾调用和热更新的支持，GCC 也开始支持 BPF 编译。同年，Cilium 1.6 发布基于 BPF 的服务发现代理（完全替换基于 iptables 的 kube-proxy）

2019年

LSM和TCP拥塞控制



直到今天，eBPF 依然是内核社区最活跃的子模块之一，还处在一个快速发展的过程中。可以说，**eBPF 开启的创新才刚刚开始**，在未来我们会看到更多的创新案例。正是为了确保每个 eBPF 学习者不掉队，我们把这门课设计成了动态发布的形式，带你随时跟踪这些最新的发展和案例。

了解了 eBPF 的诞生过程后，还有一点需要你留意：在内核社区的开发讨论中，通常还是使用 BPF 这个缩略词，而在很多应用的文档中可能会倾向使用 eBPF。其实它们的含义是一样的，都是指扩展版的 BPF。

eBPF 是怎么工作的？

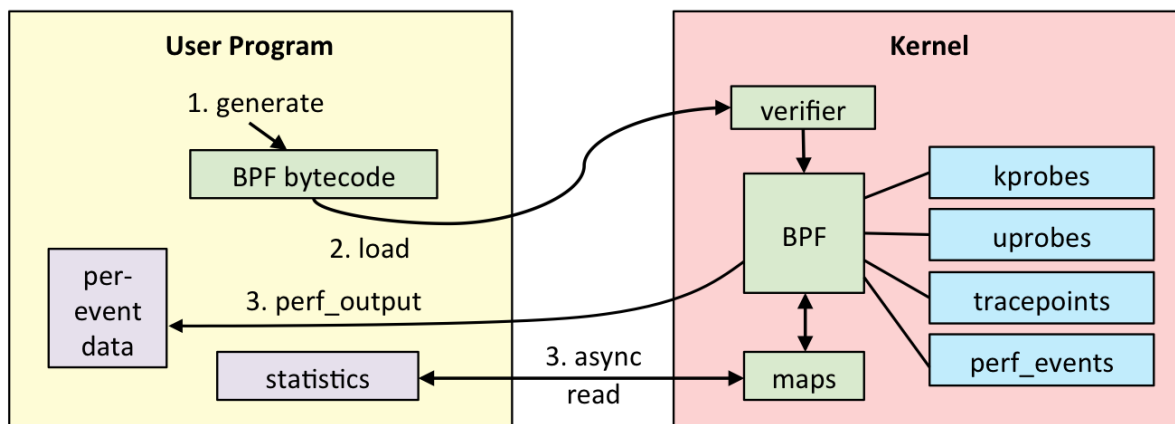
eBPF 程序并不像常规的线程那样，启动后就一直运行在那里，它需要事件触发后才会执行。这些事件包括系统调用、内核跟踪点、内核函数和用户态函数的调用退出、网络事件，等等。借助于强大的内核态插桩（kprobe）和用户态插桩（uprobe），eBPF 程序几乎可以在内核和应用的任意位置进行插桩。

看到这个令人惊叹的能力，你一定有疑问：这会不会像内核模块一样，一个异常的 eBPF 程序就会损坏整个内核的稳定性呢？其实，**确保安全和稳定一直都是 eBPF 的首要任务**，

不安全的 eBPF 程序根本就不会提交到内核虚拟机中执行。

Linux 内核是如何实现 eBPF 程序的安全和稳定的呢？其实很简单，我带你看看 eBPF 程序的执行过程，你就明白了。

如下图（图片来自 brendangregg.com）所示，通常我们借助 [LLVM](https://llvm.org) 把编写的 eBPF 程序转换为 BPF 字节码，然后再通过 bpf 系统调用提交给内核执行。内核在接受 BPF 字节码之前，会首先通过验证器对字节码进行校验，只有校验通过的 BPF 字节码才会提交到即时编译器执行。



eBPF 程序执行过程

如果 BPF 字节码中包含了不安全的操作，验证器会直接拒绝 BPF 程序的执行。比如，下面就是一些典型的验证过程：

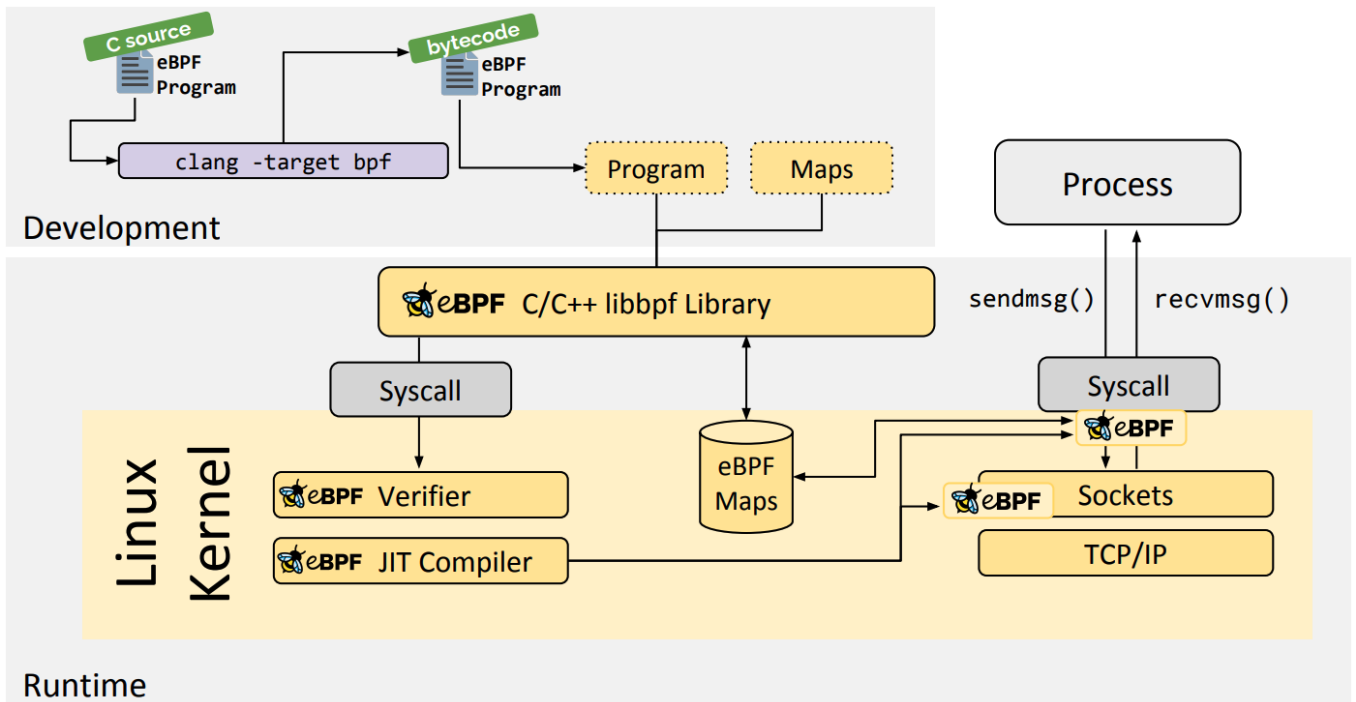
只有特权进程才可以执行 bpf 系统调用；

BPF 程序不能包含无限循环；

BPF 程序不能导致内核崩溃；

BPF 程序必须在有限时间内完成。

BPF 程序可以利用 BPF 映射（map）进行存储，而用户程序通常也需要通过 BPF 映射同运行在内核中的 BPF 程序进行交互。如下图（图片来自 ebpf.io）所示，在性能观测中，BPF 程序收集内核运行状态存储在映射中，用户程序再从映射中读出这些状态。



BPF 映射

可以看到，eBPF 程序的运行需要历经编译、加载、验证和内核态执行等过程，而用户态程序则需要借助 BPF 映射来获取内核态 eBPF 程序的运行状态。

eBPF 是万能的吗？

看到这里，你是不是因为 eBPF 在扩展内核功能上的强大能力而兴奋不已？我猜你已经迫不及待想要体验一下了。不过，在你体验之前，我还要提醒你一点：eBPF 并不是万能的，它也有很多的局限性。下面是一些最常见的 eBPF 限制：

eBPF 程序必须被验证器校验通过后才能执行，且不能包含无法到达的指令；

eBPF 程序不能随意调用内核函数，只能调用在 API 中定义的辅助函数；

eBPF 程序栈空间最多只有 512 字节，想要更大的存储，就必须借助映射存储；

在内核 5.2 之前，eBPF 字节码最多只支持 4096 条指令，而 5.2 内核把这个限制提高到了 100 万条；

由于内核的快速变化，在不同版本内核中运行时，需要访问内核数据结构的 eBPF 程序很可能需要调整源码，并重新编译。

此外，虽然 Linux 内核很早就已经支持了 eBPF，但很多新特性都是在 4.x 版本中逐步增加的，具体你可以看下 [这个链接](#)。所以，想要稳定运行 eBPF 程序，**内核版本至少需要 4.9 或者更新**。而在开发和学习 eBPF 时，**为了体验最新的 eBPF 特性，我推荐使用更新的 5.x**

内核。在这门课后面的内容中，我还会给你详细讲解开发环境的搭建步骤，以及推荐的 Linux 发行版。

小结

今天，我带你一起探索了 eBPF 技术的发展历程，并梳理了 eBPF 的工作原理。

eBPF 是从 BPF 技术扩展而来的。BPF 出现后，一直都是网络数据包过滤的核心，但直到 eBPF 诞生前，BPF 都仅用于包过滤这个场景中。eBPF 的诞生是 BPF 技术的一个转折点，使它的应用范围逐步从包过滤扩展到内核函数、用户函数、跟踪点、性能事件、安全控制等全新的领域中。而这也进一步催生了 Cilium、Kraton、Falco 等一大批基于 eBPF 构建的网络和安全解决方案，形成了繁荣的 eBPF 生态。

eBPF 程序以内核事件触发的方式运行，并且其运行过程包括编译、加载、验证和内核态执行等。为了保护内核的安全和稳定，如果编译后 BPF 字节码中包含了不安全的操作，验证阶段会直接拒绝 BPF 程序的执行。

不过，需要提醒你的是：为了确保安全和稳定，eBPF 程序也有很多限制，这是你在后续的学习过程中需要特别留心的。

思考题

在这一讲的最后，想和你交流的问题是：在之前的学习和工作中，你有没有使用过 eBPF 呢？如果使用过，你又用 eBPF 解决过哪些实际的问题呢？期待你在留言区分享，并和我交流讨论。

今天的内容到这里就结束了，欢迎把它分享给你的同事和朋友，我们下一讲见。

分享给需要的人，Ta 订阅超级会员，你将得 50 元

Ta 单独订阅本课程，你将得 20 元

 生成海报并分享

 赞 7  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 开篇词 | 想要洞悉系统底层的黑盒？先掌握 eBPF！

下一篇 02 | 先利其器：如何高效学习 eBPF？

精选留言 (9)

写留言

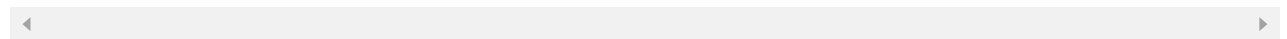


大卫李

2022-01-17

非常高兴看到eBPF技术能在极客时间开学习专栏了，算是中文圈的一个里程碑！本人也在一直学习bpf技术（也简单写了发展史：<https://davidlovezoe.club/bpf>），希望跟倪老师及大家共勉。

作者回复：欢迎在留言区跟同学们分享学习中的问题和经验。



共 3 条评论 >

11

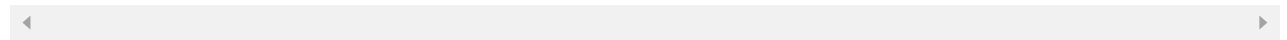


Unknown

2022-01-17

bcc tools 工具都很好用，比如Diffie-Hellman密钥交换算法使得就算有key都无法解密，不过如果通过bcc tools中的sslsniff能直接查看了，对调试来说挺方便的

作者回复：嗯嗯，是的，谢谢分享。欢迎在学习过程中分享更多的实践经验！



2

2



Damoncui

2022-01-18

好激动！目前正在看翻译版的《bpf之巅》，配合食用简直快乐到不行~英文版吃起来太痛苦.....

选对课程、教程基本节约了一半时间！

...

展开 ∨

作者回复：嗯嗯，这本书也不错，我们一起加油！



1

**余生**

2022-01-18

极客上特别喜欢的几位大咖老师，倪朋飞、陶辉、刘超、彭东！

作者回复: 谢谢对专栏的支持，我们一起加油！



1

**坚**

2022-01-18

倪老师好，我是做嵌入式开发，需要自己配置内核，在搭建bpf的使用环境的时候可以也提一下需要开启内核哪些配置吗？

作者回复: 嗯嗯，我们第03讲会详细介绍开发环境的搭建的

共 2 条评论 >



1

**我要收购腾讯**

2022-01-21

用过 bcc profile + crd 的方式给公司内部的k8s控制台开发过一键生成火焰图的功能, (全套的可观测性部署不太适合开发测试用集群)

作者回复: 谢谢分享！欢迎在学习过程中分享更多的学习和实践经验！



1

**Ethan Liu**

2022-01-20

eBPF程序执行需要进行验证，不影响性能吗？

作者回复: 只是在加载到内核之前进行验证，加载之后就不需要了

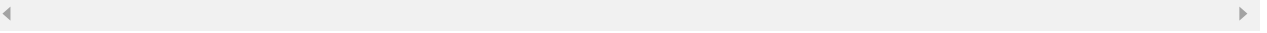
**娟**



2022-01-20

bpf很早就有一些了解，觉得那些语法挺好玩，跟着大牛一起进步，系统化了解了解ebpf。希望结束的时候能很有收获。

作者回复: 加油！

**Christopher**

2022-01-19

刚来到极客时间买的早期课程就有倪老师的Linux优化实战，看过多遍，大神必出精品！

作者回复: 感谢对专栏的支持，我们一起加油！

